## Working Draft text

# Subtest B5

#### 9.3.3.9 Binarization process for coeff\_abs\_level\_remaining

…

The variable cRiceParam is derived from cLastAbsLevel and cLastRiceParam as follows:

* If fast\_rice\_adaptation\_enabled\_flag is equal to 0, the following applies:

cRiceParam = Min( cLastRiceParam + ( cLastAbsLevel > ( 3 \* ( 1  <<  cLastRiceParam ) ) ? 1 : 0 ), 4 ) (9‑13)

* Otherwise (fast\_rice\_adaptation\_enabled\_flag is equal to 1), the following applies:
* cRiceParam is initialized as follows:

cRiceParam = Min( cLastRiceParam + ( cLastAbsLevel  >>  ( 2  +  cLastRiceParam ) ), 7 ) (9‑13)

* When this process is invoked for the first time for the current sub-block scan index i, cRiceParam is modified as follows:

cRiceParam = Max( 0, cRiceParam −  (9‑13)  
 ( transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] | | cu\_transquant\_bypass\_flag ? 1 : 2 ) )

The variable cRiceParamPrev is set to cRiceParam

…

#### 7.3.8.11 Residual coding syntax

|  |  |
| --- | --- |
| … |  |
| lastGreater1ScanPos = −1 |  |
| if( cRiceParamPrev < 4 ) { |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| if( numGreater1Flag < 8 ) { |  |
| **coeff\_abs\_level\_greater1\_flag**[ n ] | ae(v) |
| numGreater1Flag++ |  |
| if( coeff\_abs\_level\_greater1\_flag[ n ] && lastGreater1ScanPos = = −1 ) |  |
| lastGreater1ScanPos = n |  |
| } |  |
| if( lastSigScanPos = = −1 ) |  |
| lastSigScanPos = n |  |
| firstSigScanPos = n |  |
| } |  |
| } |  |
| if( cu\_transquant\_bypass\_flag  | | ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA &&   implicit\_rdpcm\_enabled\_flag && transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] &&   ( predModeIntra = = 10 | | predModeIntra = = 26 ) )  | | explicit\_rdpcm\_flag[ x0 ][ y0 ][ cIdx ] ) |  |
| signHidden = 0 |  |
| else |  |
| signHidden = ( lastSigScanPos − firstSigScanPos > 3 ) |  |
| if( lastGreater1ScanPos != −1 ) |  |
| **coeff\_abs\_level\_greater2\_flag**[ lastGreater1ScanPos ] | ae(v) |
| } |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] &&   ( !sign\_data\_hiding\_enabled\_flag | | !signHidden | | ( n != firstSigScanPos ) ) ) |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| } |  |
| numSigCoeff = 0 |  |
| sumAbsLevel = 0 |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| baseLevel = 1 + coeff\_abs\_level\_greater1\_flag[ n ] +  coeff\_abs\_level\_greater2\_flag[ n ] |  |
| if( baseLevel = = ( ( numSigCoeff < 8 ) ?   ( (n = = lastGreater1ScanPos) ? 3 : 2 ) : 1 ) ||  cRiceParamPrev > = 4 ) |  |
| **coeff\_abs\_level\_remaining**[ n ] | ae(v) |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =   ( coeff\_abs\_level\_remaining[ n ] + baseLevel ) \* ( 1 − 2 \* coeff\_sign\_flag[ n ] ) |  |
| if( sign\_data\_hiding\_enabled\_flag && signHidden ) { |  |
| sumAbsLevel += ( coeff\_abs\_level\_remaining[ n ] + baseLevel ) |  |
| if( ( n = = firstSigScanPos ) && ( ( sumAbsLevel % 2 ) = = 1 ) ) |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  −TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] |  |
| } |  |
| numSigCoeff++ |  |
| } |  |
| } |  |
| } |  |
| } |  |

NOTE – when alignment is enabled and cRiceParamPrev is greater than or equal to 4, the entropy coder is limited to coding 16 (significance map) + 8 (greater-than-one map) + 1 (greater-than-two flag) = 25 CABAC bins per coefficient group; all escape bins are available raw at the point of decoding. When alignment is enabled and cRiceParamPrev is less than 4, the entropy coder is limited to coding 25 CABAC bins and 16 EP CABAC bins per coefficient group.

# Subtest B6

#### 9.3.3.9 Binarization process for coeff\_abs\_level\_remaining

…

The variable cRiceParam is derived from cLastAbsLevel and cLastRiceParam as follows:

* If fast\_rice\_adaptation\_enabled\_flag is equal to 0, the following applies:

cRiceParam = Min( cLastRiceParam + ( cLastAbsLevel > ( 3 \* ( 1  <<  cLastRiceParam ) ) ? 1 : 0 ), 4 ) (9‑13)

* Otherwise (fast\_rice\_adaptation\_enabled\_flag is equal to 1), the following applies:
* cRiceParam is initialized as follows:

cRiceParam = Min( cLastRiceParam + ( cLastAbsLevel  >>  ( 2  +  cLastRiceParam ) ), 7 ) (9‑13)

* When this process is invoked for the first time for the current sub-block scan index i, cRiceParam is modified as follows:

cRiceParam = Max( 0, cRiceParam −  (9‑13)  
 ( transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] | | cu\_transquant\_bypass\_flag ? 1 : 2 ) )

The variable cRiceParamPrev is set to cRiceParam

…

#### 7.3.8.11 Residual coding syntax

|  |  |
| --- | --- |
| residual\_coding( x0, y0, log2TrafoSize, cIdx ) { | Descriptor |
| … |  |
| for( i = lastSubBlock; i >= 0; i− − ) { |  |
| xS = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ i ][ 0 ] |  |
| yS = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ i ][ 1 ] |  |
| if( cRiceParamPrev < 4) { |  |
| inferSbDcSigCoeffFlag = 0 |  |
| if( ( i < lastSubBlock ) && ( i > 0 ) ) { |  |
| **coded\_sub\_block\_flag**[ xS ][ yS ] | ae(v) |
| inferSbDcSigCoeffFlag = 1 |  |
| } |  |
| for( n = ( i = = lastSubBlock ) ? lastScanPos − 1 : 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( coded\_sub\_block\_flag[ xS ][ yS ] && ( n > 0 | | !inferSbDcSigCoeffFlag ) ) { |  |
| **sig\_coeff\_flag**[ xC ][ yC ] | ae(v) |
| if( sig\_coeff\_flag[ xC ][ yC ] ) |  |
| inferSbDcSigCoeffFlag = 0 |  |
| } |  |
| } |  |
| firstSigScanPos = 16 |  |
| lastSigScanPos = −1 |  |
| numGreater1Flag = 0 |  |
| lastGreater1ScanPos = −1 |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| if( numGreater1Flag < 8 ) { |  |
| **coeff\_abs\_level\_greater1\_flag**[ n ] | ae(v) |
| numGreater1Flag++ |  |
| if( coeff\_abs\_level\_greater1\_flag[ n ] && lastGreater1ScanPos = = −1 ) |  |
| lastGreater1ScanPos = n |  |
| } |  |
| if( lastSigScanPos = = −1 ) |  |
| lastSigScanPos = n |  |
| firstSigScanPos = n |  |
| } |  |
| } |  |
| if( cu\_transquant\_bypass\_flag  | | ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA &&   implicit\_rdpcm\_enabled\_flag && transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] &&   ( predModeIntra = = 10 | | predModeIntra = = 26 ) )  | | explicit\_rdpcm\_flag[ x0 ][ y0 ][ cIdx ] ) |  |
| signHidden = 0 |  |
| Else |  |
| signHidden = ( lastSigScanPos − firstSigScanPos > 3 ) |  |
| if( lastGreater1ScanPos != −1 ) |  |
| **coeff\_abs\_level\_greater2\_flag**[ lastGreater1ScanPos ] | ae(v) |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] &&   ( !sign\_data\_hiding\_enabled\_flag | | !signHidden | | ( n != firstSigScanPos ) ) ) |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| } |  |
| numSigCoeff = 0 |  |
| sumAbsLevel = 0 |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| baseLevel = 1 + coeff\_abs\_level\_greater1\_flag[ n ] +  coeff\_abs\_level\_greater2\_flag[ n ] |  |
| if( baseLevel = = ( ( numSigCoeff < 8 ) ?   ( (n = = lastGreater1ScanPos) ? 3 : 2 ) : 1 ) ) |  |
| **coeff\_abs\_level\_remaining**[ n ] | ae(v) |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =   ( coeff\_abs\_level\_remaining[ n ] + baseLevel ) \* ( 1 − 2 \* coeff\_sign\_flag[ n ] ) |  |
| if( sign\_data\_hiding\_enabled\_flag && signHidden ) { |  |
| sumAbsLevel += ( coeff\_abs\_level\_remaining[ n ] + baseLevel ) |  |
| if( ( n = = firstSigScanPos ) && ( ( sumAbsLevel % 2 ) = = 1 ) ) |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  −TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] |  |
| } |  |
| numSigCoeff++ |  |
| } |  |
| } |  |
| } else { |  |
| firstSigScanPos = 16 |  |
| lastSigScanPos = −1 |  |
| for( n = ( i = = lastSubBlock ) ? lastScanPos − 1 : 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| baseLevel = ( xC == LastSignificantCoeffX ) && ( yC == LastSignificantCoeffY ) |  |
| **coeff\_abs\_level\_remaining**[ n ] | ae(v) |
| coeff\_abs\_level\_remaining[ n ] += baseLevel |  |
| if( coeff\_abs\_level\_remaining[ n ] > 0 ) |  |
| sig\_coeff\_flag[ xC ][ yC ] = 1 |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| if( lastSigScanPos = = −1 ) |  |
| lastSigScanPos = n |  |
| firstSigScanPos = n |  |
| } |  |
| } |  |
| if( cu\_transquant\_bypass\_flag  | | ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA &&   implicit\_rdpcm\_enabled\_flag && transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] &&   ( predModeIntra = = 10 | | predModeIntra = = 26 ) )  | | explicit\_rdpcm\_flag[ x0 ][ y0 ][ cIdx ] ) |  |
| signHidden = 0 |  |
| else |  |
| signHidden = ( lastSigScanPos − firstSigScanPos > 3 ) |  |
| sumAbsLevel = 0 |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] &&   ( !sign\_data\_hiding\_enabled\_flag | | !signHidden | | ( n != firstSigScanPos ) ) ) |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =   coeff\_abs\_level\_remaining[ n ] \* ( 1 − 2 \* coeff\_sign\_flag[ n ] ) |  |
| if( sign\_data\_hiding\_enabled\_flag && signHidden ) { |  |
| sumAbsLevel += coeff\_abs\_level\_remaining[ n ] |  |
| if( ( n = = firstSigScanPos ) && ( ( sumAbsLevel % 2 ) = = 1 ) ) |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  −TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |

NOTE – when alignment is enabled and cRiceParamPrev is greater than or equal to 4, the entropy coder is limited to coding 16 (significance map) + 8 (greater-than-one map) + 1 (greater-than-two flag) = 25 CABAC bins per coefficient group; all escape bins are available raw at the point of decoding. When alignment is enabled and cRiceParamPrev is less than 4, the entropy coder is limited to coding 25 CABAC bins and 16 EP CABAC bins per coefficient group.

# Common specification changes for B5 and B6

#### 7.3.2.2 Sequence parameter set RBSP syntax

|  |  |
| --- | --- |
| seq\_parameter\_set\_rbsp( ) { | Descriptor |
| **sps\_video\_parameter\_set\_id** | u(4) |
| **…** |  |
| **bit\_depth\_luma\_minus8** | ue(v) |
| **bit\_depth\_chroma\_minus8** | ue(v) |
| **align\_cabac\_before\_bypass\_data\_flag** | u(1) |
| **log2\_max\_pic\_order\_cnt\_lsb\_minus4** | ue(v) |
| **…** | u(1) |
| rbsp\_trailing\_bits( ) |  |
| } |  |

**7.4.3.2 Sequence parameter set RBSP semantics**

…

**bit\_depth\_chroma\_minus8** specifies the bit depth of the samples of the chroma arrays BitDepthC and the value of the chroma quantization parameter range offset QpBdOffsetC as follows:

BitDepthC = 8 + bit\_depth\_chroma\_minus8 (7‑6)

QpBdOffsetC = 6 \* bit\_depth\_chroma\_minus8 (7‑7)

bit\_depth\_chroma\_minus8 shall be in the range of 0 to 6, inclusive.

**align\_cabac\_before\_bypass\_data\_flag** specifies the use of aligned bypass coding during the entropy coding process.

When align\_cabac\_before\_bypass\_data\_flag is not present, it is inferred to be equal to 0.

…

[ Note that existing profiles would be amended to include the clause “The syntax element align\_cabac\_before\_bypass\_data\_flag shall not be present in the bit stream.” ]

**9.3.4 Decoding process flow**

**9.3.4.1 General**

Inputs to this process are all bin strings of the binarization of the requested syntax element as specified in subclause 9.3.3.

Output of this process is the value of the syntax element.

This process specifies how each bin of a bin string is parsed for each syntax element. After parsing each bin, the resulting bin string is compared to all bin strings of the binarization of the syntax element and the following applies:

– If the bin string is equal to one of the bin strings, the corresponding value of the syntax element is the output.

– Otherwise (the bin string is not equal to one of the bin strings), the next bit is parsed.

While parsing each bin, the variable binIdx is incremented by 1 starting with binIdx being set equal to 0 for the first bin.

The parsing of each bin is specified by the following two ordered steps:

1. The derivation process for ctxTable, ctxIdx, ~~and~~ bypassFlag, and alignFlag as specified in subclause 9.3.4.2 is invoked with binIdx as input and ctxTable, ctxIdx, ~~and~~ bypassFlag, and alignFlag as outputs.

2. The arithmetic decoding process as specified in subclause 9.3.4.3 is invoked with ctxTable, ctxIdx, ~~and~~ bypassFlag, and alignFlag as inputs and the value of the bin as output.

**9.3.4.2 Derivation process for ctxTable, ctxIdx ~~and~~, bypassFlag and alignFlag**

**9.3.4.2.1 General**

Input to this process is the position of the current bin within the bin string, binIdx.

Outputs of this process are ctxTable, ctxIdx, ~~and~~ bypassFlag and alignFlag.

The values of ctxTable, ctxIdx, ~~and~~ bypassFlag and alignFlag are derived as follows based on the entries for binIdx of the corresponding syntax element in Table 9‑37:

* If the entry in Table 9‑37 is not equal to "bypass", "bypass\_align", "terminate", and "na", the values of binIdx are decoded by invoking the DecodeDecision process as specified in subclause 9.3.4.3.2 and the following applies:
* ctxTable is specified in Table 9‑4.
* The variable ctxInc is specified by the corresponding entry in Table 9‑37 and when more than one value is listed in Table 9‑37 for a binIdx, the assignment process for ctxInc for that binIdx is further specified in the subclauses given in parenthesis.
* The variable ctxIdxOffset is specified by the lowest value of ctxIdx in Table 9‑4 depending on the current value of initType.
* ctxIdx is set equal to the sum of ctxInc and ctxIdxOffset.
* bypass Flag is set equal to 0.
* alignFlag is set equal to 0.
* Otherwise, if the entry in Table 9‑37 is equal to "bypass", the values of binIdx are decoded by invoking the DecodeBypass process as specified in subclause 9.3.4.3.4 and the following applies:
* ctxTable is set equal to 0.
* ctxIdx is set equal to 0.
* bypassFlag is set equal to 1.
* alignFlag is set equal to 0.
* Otherwise, if the entry in Table 9-37 is equal to "bypass\_align", the values of binIdx are decoded by invoking the DecodeBypass process as specified in subclause 9.3.4.3.4 and the following applies:
  + ctxTable is set equal to 0.
  + ctxIdx is set equal to 0.
  + bypassFlag is set equal to 1.
  + If cRiceParamPrev is greater than or equal to 4, alignFlag is set equal to the value of align\_cabac\_before\_bypass\_data\_flag.
  + Otherwise (cRiceParamPrev is less than 4) alignFlag is set equal to 0.
* Otherwise, if the entry in Table 9‑37 is equal to "terminate", the values of binIdx are decoded by invoking the DecodeTerminate process as specified in subclause 9.3.4.3.5 and the following applies:
* ctxTable is set equal to 0.
* ctxIdx is set equal to 0.
* bypassFlag is set equal to 0.
* alignFlag is set equal to 0.
* Otherwise (the entry in Table 9‑37 is equal to "na"), the values of binIdx do not occur for the corresponding syntax element.

| Table 3 – Assignment of ctxInc to syntax elements with context coded bins | | | | | | |
| --- | --- | --- | --- | --- | --- | --- |
| **Syntax element** | **binIdx** | | | | | |
| **0** | **1** | **2** | **3** | **4** | **>= 5** |
| end\_of\_slice\_segment\_flag | terminate | na | na | na | na | na |
| … | … | … | … | … | … | … |
| ~~coeff\_abs\_level\_remaining[ ]~~ | ~~bypass~~ | ~~bypass~~ | ~~bypass~~ | ~~bypass~~ | ~~bypass~~ | ~~bypass~~ |
| ~~coeff\_sign\_flag[ ]~~ | ~~bypass~~ | ~~na~~ | ~~na~~ | ~~na~~ | ~~na~~ | ~~na~~ |
| coeff\_abs\_level\_remaining[ ] | bypass\_align | bypass\_align | bypass\_align | bypass\_align | bypass\_align | bypass\_align |
| coeff\_sign\_flag[ ] | bypass\_align | na | na | na | na | na |

**9.3.4.3.1 General**

Inputs to this process are ctxTable, ctxIdx, ~~and~~ bypassFlag and alignFlag, as derived in subclause 9.3.4.2, and the state variables ivlCurrRange and ivlOffset of the arithmetic decoding engine.

Output of this process is the value of the bin.

Figure 9‑5 illustrates the whole arithmetic decoding process for a single bin. For decoding the value of a bin, the context index table ctxTable and the ctxIdx are passed to the arithmetic decoding process DecodeBin( ctxTable, ctxIdx ), which is specified as follows:

– If bypassFlag is equal to 1, DecodeBypass( alignFlag ) as specified in subclause 9.3.4.3.4 is invoked.

– Otherwise, if bypassFlag is equal to 0, ctxTable is equal to 0, and ctxIdx is equal to 0, DecodeTerminate( ) as specified in subclause 9.3.4.3.5 is invoked.

– Otherwise (bypassFlag is equal to 0 and ctxTable is not equal to 0), DecodeDecision( ) as specified in subclause 9.3.4.3.2 is invoked.

~~~~



Figure 9‑5 – Overview of the arithmetic decoding process for a single bin (informative)

**9.3.4.3.4 Bypass decoding process for binary decisions**

Inputs to this process are bits from slice segment data and the variables ivlCurrRange, ~~and~~ ivlOffset and alignFlag.

Outputs of this process are the updated variables ivlOffset and ivlCurrRange and the decoded value binVal.

The bypass decoding process is invoked when bypassFlag is equal to 1. Figure 9‑8 shows a flowchart of the corresponding process.

If alignFlag is equal to 1:

– ivlCurrRange is set to 256.

Then~~First~~, the value of ivlOffset is doubled, i.e. left-shifted by 1 and a single bit is shifted into ivlOffset by using read\_bits( 1 ). Then, the value of ivlOffset is compared to the value of ivlCurrRange and further steps are specified as follows:

– If ivlOffset is greater than or equal to ivlCurrRange, the variable binVal is set equal to 1 and ivlOffset is decremented by ivlCurrRange.

– Otherwise (ivlOffset is less than ivlCurrRange), the variable binVal is set equal to 0*.*

The bitstream shall not contain data that result in a value of ivlOffset being greater than or equal to ivlCurrRange upon completion of this process.

NOTE – when ivlCurrRange is 256, the process can be simplified: ivlOffset and the bit-stream can be considered as a shift register, and binVal is the most-significant-bit-but-1 (the most significant bit is always 0 due to above restriction of ivlOffset being less than ivlCurrRange).

~~~~



Figure 9‑8 – Flowchart of bypass decoding process

**9.3.5.5 Bypass encoding process for binary decisions (informative)**

This subclause does not form an integral part of this Specification.

Inputs to this process are the variables binVal, alignFlag, ivlLow, ivlCurrRange, bitsOutstanding, and BinCountsInNalUnits.

Output of this process is a bit written to the RBSP and the updated variables ivlLow, bitsOutstanding, and BinCountsInNalUnits.

This encoding process applies to all binary decisions with bypassFlag equal to 1. Renormalization is included in the specification of this process as given in Figure 9‑13.

~~~~



Figure 9‑13 – Flowchart of encoding bypass

# Subtest A2

#### 9.3.3.9 Binarization process for coeff\_abs\_level\_remaining

…

The variable cRiceParam is derived from cLastAbsLevel and cLastRiceParam as follows:

* If fast\_rice\_adaptation\_enabled\_flag is equal to 0, the following applies:

cRiceParam = Min( cLastRiceParam + ( cLastAbsLevel > ( 3 \* ( 1  <<  cLastRiceParam ) ) ? 1 : 0 ), 4 ) (9‑13)

* Otherwise (fast\_rice\_adaptation\_enabled\_flag is equal to 1), the following applies:
* cRiceParam is initialized as follows:

cRiceParam = Min( cLastRiceParam + ( cLastAbsLevel  >>  ( 2  +  cLastRiceParam ) ), 7 ) (9‑13)

* When this process is invoked for the first time for the current sub-block scan index i, cRiceParam is modified as follows:

cRiceParam = Max( 0, cRiceParam −  (9‑13)  
 ( transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] | | cu\_transquant\_bypass\_flag ? 1 : 2 ) )

The variable cRiceParamPrev is set to cRiceParam

…

#### 7.3.8.11 Residual coding syntax

|  |  |
| --- | --- |
| residual\_coding( x0, y0, log2TrafoSize, cIdx ) { | Descriptor |
| … |  |
| for( i = lastSubBlock; i >= 0; i− − ) { |  |
| xS = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ i ][ 0 ] |  |
| yS = ScanOrder[ log2TrafoSize − 2 ][ scanIdx ][ i ][ 1 ] |  |
| if( cRiceParamPrev < 4) { |  |
| inferSbDcSigCoeffFlag = 0 |  |
| if( ( i < lastSubBlock ) && ( i > 0 ) ) { |  |
| **coded\_sub\_block\_flag**[ xS ][ yS ] | ae(v) |
| inferSbDcSigCoeffFlag = 1 |  |
| } |  |
| for( n = ( i = = lastSubBlock ) ? lastScanPos − 1 : 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( coded\_sub\_block\_flag[ xS ][ yS ] && ( n > 0 | | !inferSbDcSigCoeffFlag ) ) { |  |
| **sig\_coeff\_flag**[ xC ][ yC ] | ae(v) |
| if( sig\_coeff\_flag[ xC ][ yC ] ) |  |
| inferSbDcSigCoeffFlag = 0 |  |
| } |  |
| } |  |
| firstSigScanPos = 16 |  |
| lastSigScanPos = −1 |  |
| numGreater1Flag = 0 |  |
| lastGreater1ScanPos = −1 |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| if( numGreater1Flag < 8 ) { |  |
| **coeff\_abs\_level\_greater1\_flag**[ n ] | ae(v) |
| numGreater1Flag++ |  |
| if( coeff\_abs\_level\_greater1\_flag[ n ] && lastGreater1ScanPos = = −1 ) |  |
| lastGreater1ScanPos = n |  |
| } |  |
| if( lastSigScanPos = = −1 ) |  |
| lastSigScanPos = n |  |
| firstSigScanPos = n |  |
| } |  |
| } |  |
| if( cu\_transquant\_bypass\_flag  | | ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA &&   implicit\_rdpcm\_enabled\_flag && transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] &&   ( predModeIntra = = 10 | | predModeIntra = = 26 ) )  | | explicit\_rdpcm\_flag[ x0 ][ y0 ][ cIdx ] ) |  |
| signHidden = 0 |  |
| else |  |
| signHidden = ( lastSigScanPos − firstSigScanPos > 3 ) |  |
| if( lastGreater1ScanPos != −1 ) |  |
| **coeff\_abs\_level\_greater2\_flag**[ lastGreater1ScanPos ] | ae(v) |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] &&   ( !sign\_data\_hiding\_enabled\_flag | | !signHidden | | ( n != firstSigScanPos ) ) ) |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| } |  |
| numSigCoeff = 0 |  |
| sumAbsLevel = 0 |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| baseLevel = 1 + coeff\_abs\_level\_greater1\_flag[ n ] +  coeff\_abs\_level\_greater2\_flag[ n ] |  |
| if( baseLevel = = ( ( numSigCoeff < 8 ) ?   ( (n = = lastGreater1ScanPos) ? 3 : 2 ) : 1 ) ) |  |
| **coeff\_abs\_level\_remaining**[ n ] | ae(v) |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =   ( coeff\_abs\_level\_remaining[ n ] + baseLevel ) \* ( 1 − 2 \* coeff\_sign\_flag[ n ] ) |  |
| if( sign\_data\_hiding\_enabled\_flag && signHidden ) { |  |
| sumAbsLevel += ( coeff\_abs\_level\_remaining[ n ] + baseLevel ) |  |
| if( ( n = = firstSigScanPos ) && ( ( sumAbsLevel % 2 ) = = 1 ) ) |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  −TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] |  |
| } |  |
| numSigCoeff++ |  |
| } |  |
| } |  |
| } else { |  |
| firstSigScanPos = 16 |  |
| lastSigScanPos = −1 |  |
| for( n = ( i = = lastSubBlock ) ? lastScanPos − 1 : 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| baseLevel = ( xC == LastSignificantCoeffX ) && ( yC == LastSignificantCoeffY ) |  |
| **coeff\_abs\_level\_remaining**[ n ] | ae(v) |
| coeff\_abs\_level\_remaining[ n ] += baseLevel |  |
| if( coeff\_abs\_level\_remaining[ n ] > 0 ) |  |
| sig\_coeff\_flag[ xC ][ yC ] = 1 |  |
| if( sig\_coeff\_flag[ xC ][ yC ] ) { |  |
| if( lastSigScanPos = = −1 ) |  |
| lastSigScanPos = n |  |
| firstSigScanPos = n |  |
| } |  |
| } |  |
| if( cu\_transquant\_bypass\_flag  | | ( CuPredMode[ x0 ][ y0 ] = = MODE\_INTRA &&   implicit\_rdpcm\_enabled\_flag && transform\_skip\_flag[ x0 ][ y0 ][ cIdx ] &&   ( predModeIntra = = 10 | | predModeIntra = = 26 ) )  | | explicit\_rdpcm\_flag[ x0 ][ y0 ][ cIdx ] ) |  |
| signHidden = 0 |  |
| else |  |
| signHidden = ( lastSigScanPos − firstSigScanPos > 3 ) |  |
| sumAbsLevel = 0 |  |
| for( n = 15; n >= 0; n− − ) { |  |
| xC = ( xS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 0 ] |  |
| yC = ( yS << 2 ) + ScanOrder[ 2 ][ scanIdx ][ n ][ 1 ] |  |
| if( sig\_coeff\_flag[ xC ][ yC ] &&   ( !sign\_data\_hiding\_enabled\_flag | | !signHidden | | ( n != firstSigScanPos ) ) ) |  |
| **coeff\_sign\_flag**[ n ] | ae(v) |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =   coeff\_abs\_level\_remaining[ n ] \* ( 1 − 2 \* coeff\_sign\_flag[ n ] ) |  |
| if( sign\_data\_hiding\_enabled\_flag && signHidden ) { |  |
| sumAbsLevel += coeff\_abs\_level\_remaining[ n ] |  |
| if( ( n = = firstSigScanPos ) && ( ( sumAbsLevel % 2 ) = = 1 ) ) |  |
| TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] =  −TransCoeffLevel[ x0 ][ y0 ][ cIdx ][ xC ][ yC ] |  |
| } |  |
| } |  |
| } |  |
| } |  |
| } |  |